情報理工学部　SN コース 2 回
セキュリティ・ネットワーク学実験 2
課題 7 レポート

2600200443-6
Yamashita Kyohei
山下 恭平

November 1 2021

1　概要

BLE(Bluetooth Low Energy) を用いて Raspberry Pi とアンドロイドスマートフォン間で通信を行い、独自の IoT デバイスを作成する。

2　外部仕様

2.1　開発対象の使い方に関する説明

Raspberry Pi にはセンサとして超音波レンジャーが接続されており、測定された距離によってアクチュエータである、ディスプレイの色が変更されるようになっている。また、スマートフォンには、センサが測定した値がグラフとして表示されるようになっている。

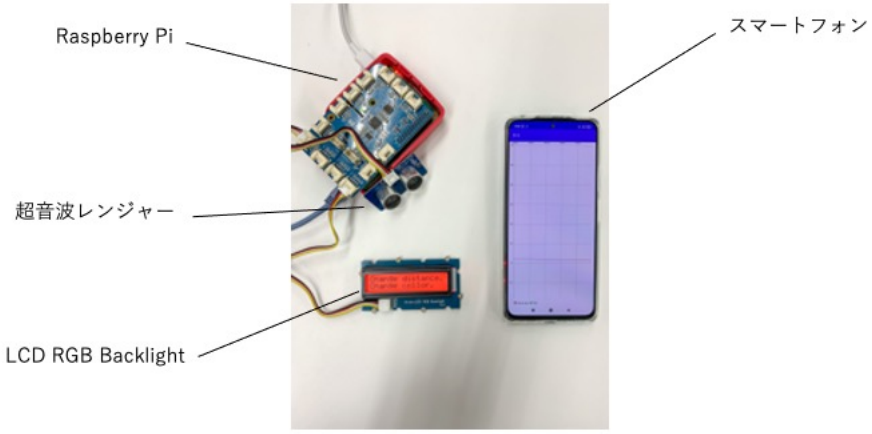

図 1　開発したデバイス

2.2　開発対象を構成するハードウェアと、その主な仕様

Raspberry Pi には Grove Starter Kit のシールドを利用し、各センサやアクチュエータを接続した。また、Raspberry Pi はセンサから情報の取得、アクチュエータの制御など、デバイスの主要部分ほとんどを担っている。超音波レンジャーは超音波により距離を測定しており、精度は 1cm 単位で測定可能である。LCD RGB Backlight は背景色を赤緑青それぞれを 256 段階で調節可能なディスプレイであり、任意の色、文字を表示することができる。スマートフォンはセントラルとして使用しており、測定した値をもとにディスプレイの色を決定する演算をおこなっている。以下の表に全てのハードウェアをまとめる。

表 1　ハードウェア一覧

機器一覧	仕様/情報
Raspberry Pi	センサ、アクチュエータを接続しそれらの制御を行う。ペリフェラルとして使用。
超音波レンジャー	超音波を送り、物体からのエコーを受信し、距離を測定。
LCD RGB Backlight	RGB バックライトを搭載したディスプレイ。
スマートフォン	Android スマートフォン。セントラルとして使用。

2.3　ハードウェアやソフトウェアが担当する機能と、機能同士の関連

まず、超音波レンジャーから距離を測定し、その値を Raspberry Pi で取得、Raspberry Pi からスマートフォンへ向けてデータを送信、スマートフォンから送られてきたデータをディスプレイに出力まで、これらがが Raspberry Pi が担う役割である。スマートフォンは測定結果を受信したらそれをグラフへ表示、その結果をもとに送信するデータの作成、データの送信を担う。以下の図はそれらの構成をまとめたものである。

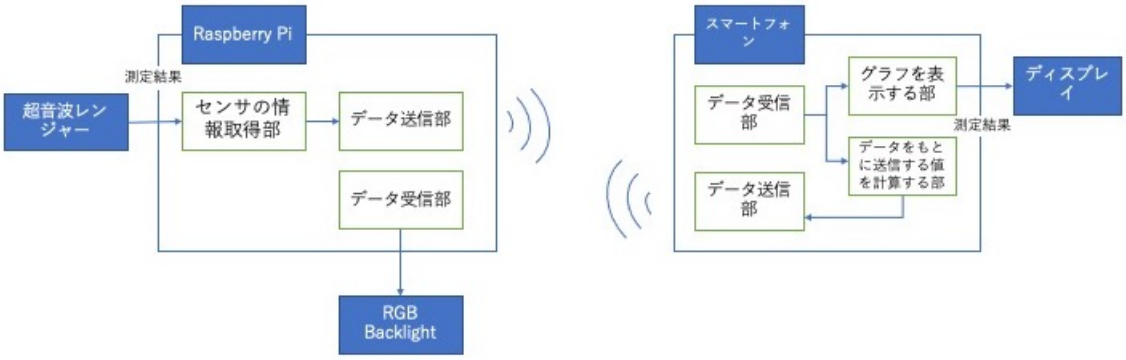

図 2　機能構成図

2.4　開発に用いたプログラミング言語と開発環境

今回の実験では Raspberry Pi の開発に Jupyter Notebook を使用し、Python にてコーディングを行った。スマートフォンの開発には Android Studio を使用しプログラミング言語として Java を使用した。以下の表は、開発環境をまとめたものである。

表 2　まとめ表

開発環境	
OS	macOS Monterey
使用したアプリケーション	Jupyter Notebook , Android Studio
使用したプログラミング言語	Python , Java

3　内部仕様

3.1　各ハードウェアが備えるソフトウェアの詳細な設計

Raspberry Pi 側では、スマートフォンへデータを送信する関数 notify やアクチュエータの設定を行う setText,setRGB、またセンサの制御として ranger,ultrasonicRead を使用した。
スマートフォン側では、受信データと電波強度の両方を格納するための配列 values[] を使用し、また、受信データを基に送信データを作成する DecideControlParameter や、これらの実行や送信データの設定などを行う onCharacteristicChanged などを使用した。以下の表 3,4 は開発デバイスごとに主な設計を表にまとめたものである。

表 3　Raspberry Pi およびセンサ、アクチュエータ

関数名/クラス名/変数名	説明
Peripheral	BLE 通信を管理するクラス、広告パケットの送信などに使用。
notify	セントラル (スマートフォン) へデータを送信する関数、センサの値を送信。
setText()	ディスプレイに表示する文字を設定。
setRGB(int , int ,int)	ディスプレイの色を設定。赤、青、緑の三色を 0 から 255 の 256 段階で調節可能。
ultrasonicRead(portnumber)	指定したポート番号から超音波レンジャーの値を取得。int 型の戻り値。
ranger	超音波レンジャーを接続するポート番号を指定。int 型。

表 4　Java

関数名/クラス名/変数名	説明
values[]	values[0]:ペリフェラルから受信したデータを格納。int 型。
	values[1]:ペリフェラルから広告パケットを受信した時の電波強度を格納。int 型。
DecideControlParameter(int)	引数 (受信データ) を基に送信するデータを作成、int 型の戻り値。
onCharacteristicChanged	values[1] へ電波強度を格納、DecideControlParameter の実行、送信データの設定を行う。

3.2　各ハードウェアが備えるソフトウェアにおける処理の流れ

Raspberry Pi 側では接続が確立したら、センサの値を取得、その値をスマートフォンへ送信、データを受信、受信データを基にディスプレイカラーを変更といった流れを無限ループで回している。スマートフォン側では、データを受信、データをグラフへ出力、送信するデータの作成、データの送信といった流れを同様に無縁ループで回すことで、双方での連続的な通信を確立している。以下は、それぞれのデバイスについてのフ

ローチャートである。

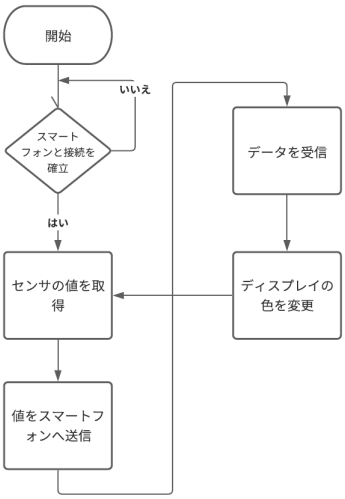

図 3　Raspberry Pi

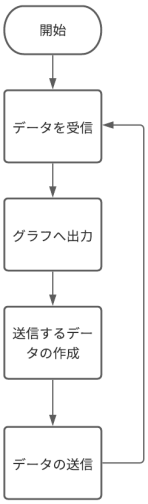

図 4　スマートフォン

4　実行例

実験は、超音波レンジャーを固定し、箱を前に置き、この箱の距離を徐々に遠ざけていくことで事件を行った。実験の結果距離が離れていくにつれて、ディスプレイの色はより緑いろへと近づき、逆に距離が近い時は赤色を示した。実際の測定結果、ディスプレイの様子および実験の様子を以下の図 5,6,7 に示す。

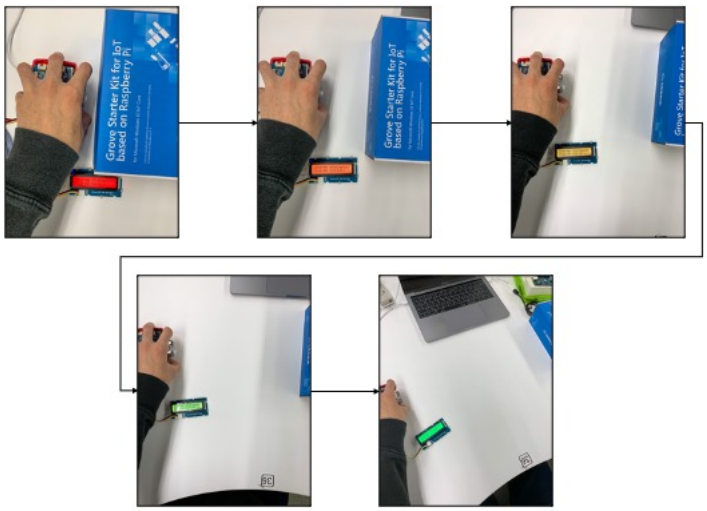

図 5　実行例 1

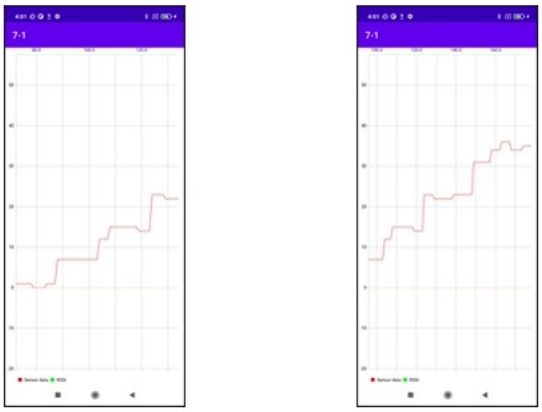

図 6　実行例 2

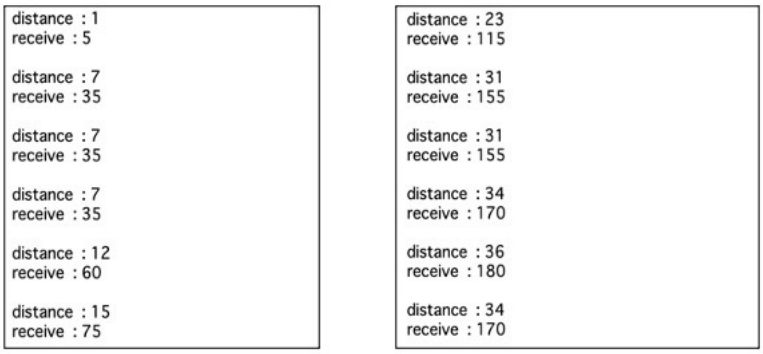

図 7　実行例 3

5　ソースコード

Raspberry Pi 側では 104 行目で超音波レンジャーのポートを設定しており、116 行目でセンサの値を取得、122 行目でデータをスマートフォンへ送信し、128 行目でスマートフォンからデータを受信、そして 137 行目でディスプレイの色を変更している。
スマートフォン側は DecideControlParameter にて、受信データを基に送信するデータを作成している。

Listing 1　7-1.py

```
1 from pybleno import *
2 from grovepi import *
3 from grove_rgb_lcd import *
4 import time
5
```

```
6  DEVICE_NAME = 'SecNet2_4436'
7  SERVICE_UUID = '19B10010-E8F2-537E-4F6C-D104768A1214'
8  WRITE_CHARACTERISTIC_UUID = '19B10011-E8F2-537E-4F6C-D104768A1214'
9  NOTIFY_CHARACTERISTIC_UUID = '19B10012-E8F2-537E-4F6C-D104768A1214'
10
11 class WriteCharacteristic(Characteristic):
12
13     def __init__(self):
14         Characteristic.__init__(self, {
15             'uuid': WRITE_CHARACTERISTIC_UUID,
16             'properties': ['write'],
17             'value': None
18         })
19
20         self._value = str(0)
21         self._updateValueCallback = None
22
23     def onWriteRequest(self, data, offset, withoutResponse, callback):
24         self._value = data.decode();
25         callback(result=Characteristic.RESULT_SUCCESS)
26
27
28 class NotifyCharacteristic(Characteristic):
29
30     def __init__(self):
31         Characteristic.__init__(self, {
32             'uuid': NOTIFY_CHARACTERISTIC_UUID,
33             'properties': ['read', 'notify'],
34             'value': None
35         })
36
37         self._value = str(0).encode()
38         self._updateValueCallback = None
39
40     def onSubscribe(self, maxValueSize, updateValueCallback):
41         print('NotifyCharacteristic - onSubscribe')
42
43         self._updateValueCallback = updateValueCallback
44
45     def onUnsubscribe(self):
46         print('NotifyCharacteristic - onUnsubscribe')
47
48         self._updateValueCallback = None
49
50
51 class Peripheral():
52     def __init__(self):
```

```
53          self.bleno = Bleno()
54          self.writeCharacteristic = WriteCharacteristic()
55          self.notifyCharacteristic = NotifyCharacteristic()
56          self.SERVICE_UUID = SERVICE_UUID
57
58      def onStateChange(self, state):
59          print('on -> stateChange: ' + state)
60
61          if (state == 'poweredOn'):
62              self.bleno.startAdvertising(name=DEVICE_NAME, service_uuids=[self.
                    SERVICE_UUID])
63          else:
64              self.bleno.stopAdvertising()
65
66      def onAdvertisingStart(self, error):
67          print('on -> advertisingStart: ' + ('error ' + error if error else 'success'))
68
69          if not error:
70              self.bleno.setServices([
71                  BlenoPrimaryService({
72                      'uuid': self.SERVICE_UUID,
73                      'characteristics': [
74                          self.writeCharacteristic,
75                          self.notifyCharacteristic
76                      ]
77                  })
78              ])
79
80      def advertise(self):
81          self.bleno.on('stateChange', self.onStateChange)
82          self.bleno.on('advertisingStart', self.onAdvertisingStart)
83          self.bleno.start()
84
85  def notify(data, characteristic):
86      if characteristic._updateValueCallback:
87  # characteristic._value = str(data)
88          characteristic._value = data
89
90          notificationBytes = str(characteristic._value).encode()
91          characteristic._updateValueCallback(data=notificationBytes)
92
93
94  def main():
95      # Peripherarl用クラスの初期化
96      peripheral = Peripheral()
97      # 広告パケットの送信開始
98      peripheral.advertise()
```

```
99      # Write/Notify通信用のCharacteristicを取得
100     writeCharacteristic = peripheral.writeCharacteristic
101     notifyCharacteristic = peripheral.notifyCharacteristic
102
103     ranger = 4
104     pinMode(ranger,"INPUT")
105
106     # Centralと接続し、データ通信の準備が整うまで待つ
107     while True:
108         if notifyCharacteristic._updateValueCallback:
109             break
110
111     setText("Change distance.\nChange collor.")
112
113     # Centralとの通信を繰り返し実行
114     while True:
115
116         sensor_value = ultrasonicRead(ranger)
117
118         # 送信するデータ(文字列)を確認
119         print('distance : ' + str(sensor_value))
120
121         # データ(文字列)をCentralへ送信
122         notify(data=str(sensor_value), characteristic=notifyCharacteristic)
123
124
125         time.sleep(0.5) # 500ミリ秒待機
126
127         # データ(文字列)をCentralから受信
128         recv_value = writeCharacteristic._value
129
130
131         # 受信したデータ(文字列)を確認
132         print('receive : ' + recv_value )
133
134         print('')
135
136
137         setRGB(255 - int(recv_value),int(recv_value),0)
138
139
140         time.sleep(2) # 500ミリ秒待機
141
142 if __name__ == '__main__':
143     main()
```

Listing 2　Java.py

```
1     package com.example.a7_1;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6
7  import androidx.core.app.ActivityCompat;
8
9  import android.Manifest;
10 import android.bluetooth.BluetoothAdapter;
11 import android.bluetooth.BluetoothGatt;
12 import android.bluetooth.BluetoothGattCallback;
13 import android.bluetooth.BluetoothGattCharacteristic;
14 import android.bluetooth.BluetoothGattDescriptor;
15 import android.bluetooth.BluetoothGattService;
16 import android.bluetooth.BluetoothManager;
17 import android.bluetooth.BluetoothProfile;
18 import android.bluetooth.le.BluetoothLeScanner;
19 import android.bluetooth.le.ScanCallback;
20 import android.bluetooth.le.ScanResult;
21 import android.content.Context;
22 import android.content.pm.PackageManager;
23 import android.graphics.Color;
24 import android.util.Log;
25 import android.view.WindowManager;
26
27 import com.github.mikephil.charting.charts.LineChart;
28 import com.github.mikephil.charting.components.AxisBase;
29 import com.github.mikephil.charting.components.XAxis;
30 import com.github.mikephil.charting.components.YAxis;
31 import com.github.mikephil.charting.data.Entry;
32 import com.github.mikephil.charting.data.LineData;
33 import com.github.mikephil.charting.data.LineDataSet;
34 import com.github.mikephil.charting.formatter.IAxisValueFormatter;
35 import com.github.mikephil.charting.utils.Transformer;
36 import com.github.mikephil.charting.utils.ViewPortHandler;
37
38 import java.util.UUID;
39
40 public class MainActivity extends AppCompatActivity {
41     private LineChart mLineChart;
42
43     // 接続対象となるペリフェラルの名前(XXXXは学籍番号の下4桁)
44     private static final String PERIPHERAL_NAME = "SecNet2_4436";
45
46     // 接続対象となるサービスのUUID.
47     private static final String SERVICE_UUID = "19B10010-E8F2-537E-4F6C-D104768A1214";
```

```
48
49      // 接続対象となるキャラクタリスティックのUUID.
50      private static final String CHAR_WRITE_UUID = "19B10011-E8F2-537E-4F6C-D104768A1214
            ";
51      private static final String CHAR_NOTIFY_UUID = "19B10012-E8F2-537E-4F6C-D104768A1214
            ";
52
53      // キャラクタリスティック設定UUID(固定値).
54      private static final String CHARACTERISTIC_CONFIG_UUID =
            "00002902-0000-1000-8000-00805f9b34fb";
55
56      // ペリフェラルと接続しない: 0, ペリフェラルと接続する: 1
57      private int flag_connect = 1;
58
59      // グラフ表示用の変数
60      private int num;
61      private float[] values;
62      private String[] labels; // データのラベルを格納する配列
63      private int[] colors; // グラフにプロットする点の色を格納する配列
64      private float max, min;
65
66      // 値をプロットするx座標
67      private float count = 0;
68
69      // BLEで利用するクラス群
70      private BluetoothManager mBleManager;
71      private BluetoothAdapter mBleAdapter;
72      private BluetoothLeScanner mBleScanner;
73      private BluetoothGatt mBleGatt;
74      private BluetoothGattCharacteristic notifyCharacteristic, writeCharacteristic;
75
76      @Override
77      protected void onCreate(Bundle savedInstanceState) {
78          super.onCreate(savedInstanceState);
79          setContentView(R.layout.activity_main);
80
81          // アプリ実行中はスリープしない
82          getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
83
84          num = 2;
85          values = new float[num];
86          labels = new String[num];
87          colors = new int[num];
88
89          for(int i=0; i<num; i++){
90              values[i] = 0;
91          }
```

```
92
93          labels[0] = "Sensor data";
94          labels[1] = "RSSI";
95
96
97          colors[0] = Color.rgb(0xFF, 0x00, 0x00); // 赤
98          colors[1] = Color.rgb(0x00, 0xFF, 0x00); // 緑
99          // colors[2] = Color.rgb(0x00, 0x00, 0xFF); // 青
100
101         max = 120;
102         min = -50;
103
104         // グラフViewを初期化する
105         initChart();
106
107         // Bluetoothの使用準備.
108         mBleManager = (BluetoothManager) getSystemService(Context.BLUETOOTH_SERVICE);
109         mBleAdapter = mBleManager.getAdapter();
110
111         // 一定間隔でグラフをアップデートする
112         new Thread(new Runnable() {
113             @Override
114             public void run() {
115                 while (true) {
116                     updateGraph();
117                     try {
118                         Thread.sleep(500);
119                     } catch (Exception e) {
120                         Log.e("Test", "例外出力", e);
121                     }
122                 }
123             }
124         }).start();
125     }
126
127     @Override
128     protected void onResume() {
129         super.onResume();
130
131         // ************************************************* //
132         // Bluetooth関連の処理 //
133         // ************************************************* //
134
135         // 位置情報の利用許可を利用者に求める (BLEも位置情報として利用できるため，許可が必要)
136         if (ActivityCompat.checkSelfPermission(this, Manifest.permission.
                ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
137             ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.
```

```
                ACCESS_FINE_LOCATION}, 1);
138
139             return;
140         }
141
142         // BLEが使用可能なら、通信相手をスキャン
143         if ((mBleAdapter != null) || (mBleAdapter.isEnabled())) {
144             mBleScanner = mBleAdapter.getBluetoothLeScanner();
145             mBleScanner.startScan(scanCallback);
146         }
147     }
148
149     private int DecideControlParameter(int input){
150         int output;
151
152         output = 5 * input;
153
154         return output;
155     }
156
157     private final BluetoothGattCallback mGattCallback = new BluetoothGattCallback() {
158         @Override
159         public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState
                )
160         {
161             // 接続状況が変化したら実行.
162             if (newState == BluetoothProfile.STATE_CONNECTED) {
163                 // 接続に成功したらサービスを検索する.
164                 gatt.discoverServices();
165             } else if (newState == BluetoothProfile.STATE_DISCONNECTED) {
166                 // 接続が切れたらGATTを空にする.
167                 if (mBleGatt != null)
168                 {
169                     mBleGatt.close();
170                     mBleGatt = null;
171                 }
172             }
173         }
174
175         @Override
176         public void onServicesDiscovered(BluetoothGatt gatt, int status)
177         {
178             // Serviceが見つかったら実行.
179             if (status == BluetoothGatt.GATT_SUCCESS) {
180                 // UUIDが同じかどうかを確認する.
181                 BluetoothGattService service = gatt.getService(UUID.fromString(
                        SERVICE_UUID));
```

```
182                 if (service != null)
183                 {
184                     // 指定したUUIDを持つCharacteristicを確認する.
185                     notifyCharacteristic = service.getCharacteristic(UUID.fromString(
                            CHAR_NOTIFY_UUID));
186                     writeCharacteristic = service.getCharacteristic(UUID.fromString(
                            CHAR_WRITE_UUID));
187
188                     if (notifyCharacteristic != null) {
189
190                         // Service, CharacteristicのUUIDが同じならBluetoothGattを更新する.
191                         mBleGatt = gatt;
192
193                         // キャラクタリスティックが見つかったら、Notificationをリクエスト.
194                         boolean registered = mBleGatt.setCharacteristicNotification(
                                notifyCharacteristic, true);
195
196                         Log.v("BLE", "Notification beginning");
197                         // Characteristic の Notificationを有効化する.
198                         BluetoothGattDescriptor descriptor = notifyCharacteristic.
                                getDescriptor(
199                                     UUID.fromString(CHARACTERISTIC_CONFIG_UUID));
200
201                         descriptor.setValue(BluetoothGattDescriptor.
                                ENABLE_NOTIFICATION_VALUE);
202                         mBleGatt.writeDescriptor(descriptor);
203                     }
204                 }
205             }
206         }
207
208         @Override
209         public void onCharacteristicChanged(BluetoothGatt gatt,
                BluetoothGattCharacteristic characteristic)
210         {
211             byte[] recvValue;
212             byte[] sendValue;
213             final String text;
214             int input, output;
215
216             // キャラクタリスティックのUUIDをチェック(getUuidの結果が全て小文字で帰ってくるので
                    UpperCaseに変換)
217             if (CHAR_NOTIFY_UUID.equals(characteristic.getUuid().toString().toUpperCase
                    ()))
218             {
219                 recvValue = characteristic.getValue();
220                 input = Integer.parseInt(new String(recvValue)); // 受信したデータを整数値
```

```
                    に変換
221                 Log.v("BLE", "Received : " + input);
222
223                 values[0] = input;
224
225                 // ペリフェラルから受信したデータを元に、ペリフェラルへ送信するデータを算出
226                 output = DecideControlParameter(input);
227
228                 sendValue = String.valueOf(output).getBytes(); // 送信するデータを文字列に
                        変換
229
230                 writeCharacteristic.setValue(sendValue);
231                 mBleGatt.writeCharacteristic(writeCharacteristic);
232                 Log.v("BLE", "Sent : " + output);
233
234             }
235         }
236     };
237
238     private ScanCallback scanCallback = new ScanCallback(){
239         @Override
240         public void onScanResult(int callbackType, ScanResult result) {
241             super.onScanResult(callbackType, result);
242
243             // 発見したペリフェラルが接続対象と一致する場合には、Rssiを取得
244             if((result.getDevice().getName() != null) && (result.getDevice().getName().
                    equals(PERIPHERAL_NAME))){
245                 values[1] = result.getRssi();
246
247                 String text = "Found: " + result.getDevice().getName() + ", " + values
                        [1];
248                 Log.d("blelog", text);
249
250                 if(flag_connect == 1){
251                     result.getDevice().connectGatt(getApplicationContext(), false,
                            mGattCallback);
252                 }
253             }
254         }
255
256         @Override
257         public void onScanFailed(int intErrorCode) {
258             super.onScanFailed(intErrorCode);
259         }
260     };
261
262     /** グラフViewの初期化 **/
```

```
263
264     private void initChart() {
265         // 線グラフView
266         mLineChart = (LineChart) findViewById(R.id.chart_DynamicMultiLineGraph);
267
268         // グラフ説明テキストを表示するか
269         mLineChart.getDescription().setEnabled(true);
270         // グラフ説明テキストのテキスト設定
271         mLineChart.getDescription().setText("Line Chart of Sensor Data");
272         // グラフ説明テキストの文字色設定
273         mLineChart.getDescription().setTextColor(Color.BLACK);
274         // グラフ説明テキストの文字サイズ設定
275         mLineChart.getDescription().setTextSize(10f);
276         // グラフ説明テキストの表示位置設定
277         mLineChart.getDescription().setPosition(0, 0);
278
279         // グラフへのタッチジェスチャーを有効にするか
280         mLineChart.setTouchEnabled(true);
281
282         // グラフのスケーリングを有効にするか
283         mLineChart.setScaleEnabled(true);
284
285         // グラフのドラッギングを有効にするか
286         mLineChart.setDragEnabled(true);
287
288         // グラフのピンチ/ズームを有効にするか
289         mLineChart.setPinchZoom(true);
290
291         // グラフの背景色設定
292         mLineChart.setBackgroundColor(Color.WHITE);
293
294         // 空のデータをセットする
295         mLineChart.setData(new LineData());
296
297         // Y軸(左)の設定
298         // Y軸(左)の取得
299         YAxis leftYAxis = mLineChart.getAxisLeft();
300         // Y軸(左)の最大値設定
301         leftYAxis.setAxisMaximum(max);
302         // Y軸(左)の最小値設定
303         leftYAxis.setAxisMinimum(min);
304
305         // Y軸(右)の設定
306         // Y軸(右)は表示しない
307         mLineChart.getAxisRight().setEnabled(false);
308
309         // X軸の設定
```

```
310         XAxis xAxis = mLineChart.getXAxis();
311         // X軸の値表示設定
312         xAxis.setValueFormatter(new IAxisValueFormatter() {
313             @Override
314             public String getFormattedValue(float value, AxisBase axis) {
315                 if(value >= 10) {
316                     // データ20個ごとに目盛りに文字を表示
317                     if (((int) value % 20) == 0) {
318                         return Float.toString(value);
319                     }
320                 }
321                 // nullを返すと落ちるので、値を書かない場合は空文字を返す
322                 return "";
323             }
324         });
325     }
326 
327     private void updateGraph() {
328         // 線の情報を取得
329         LineData lineData = mLineChart.getData();
330         if(lineData == null) {
331             return;
332         }
333 
334         LineDataSet[] lineDataSet = new LineDataSet[num];
335 
336         for(int i = 0; i<num; i++){
337             // i番目の線を取得
338             lineDataSet[i] = (LineDataSet) lineData.getDataSetByIndex(i);
339             // i番目の線が初期化されていない場合は初期化する
340             if( lineDataSet[i] == null) {
341                 // LineDataSetオブジェクト生成
342                 lineDataSet[i] = new LineDataSet(null, labels[i]);
343                 // 線の色設定
344                 lineDataSet[i].setColor(colors[i]);
345                 // 線にプロット値の点を描画しない
346                 lineDataSet[i].setDrawCircles(false);
347                 // 線にプロット値の値テキストを描画しない
348                 lineDataSet[i].setDrawValues(false);
349                 // 線を追加
350                 lineData.addDataSet(lineDataSet[i]);
351             }
352             // i番目の線に値を追加
353             lineData.addEntry(new Entry(count, values[i]), i);
354         }
355 
356         // 値更新通知
```

```
357         mLineChart.notifyDataSetChanged();
358
359         // X軸に表示する最大のEntryの数を指定
360         mLineChart.setVisibleXRangeMaximum(100);
361
362         // オシロスコープのように古いデータを左に寄せていくように表示位置をずらす
363         mLineChart.moveViewTo(count, getVisibleYCenterValue(mLineChart), YAxis.
                AxisDependency.LEFT);
364
365         count++;
366     }
367
368     /**
369      * 表示しているY座標の中心値を返す<br>
370      * 基準のY座標は左
371      * @param lineChart 対象のLineChart
372      * @return 表示しているY座標の中心値
373      */
374     private float getVisibleYCenterValue(LineChart lineChart) {
375         Transformer transformer = lineChart.getTransformer(YAxis.AxisDependency.LEFT);
376         ViewPortHandler viewPortHandler = lineChart.getViewPortHandler();
377
378         float highestVisibleY = (float) transformer.getValuesByTouchPoint(viewPortHandler
                .contentLeft(),
379                 viewPortHandler.contentTop()).y;
380         float highestY = Math.min(lineChart.getAxisLeft().mAxisMaximum, highestVisibleY)
                ;
381
382         float lowestVisibleY = (float) transformer.getValuesByTouchPoint(viewPortHandler
                .contentLeft(),
383                 viewPortHandler.contentBottom()).y;
384         float lowestY = Math.max(lineChart.getAxisLeft().mAxisMinimum, lowestVisibleY);
385
386         return highestY - (Math.abs(highestY - lowestY) / 2);
387     }
388 }
```

6　センサネットワーク実験に対する感想

センサやアクチュエータを初めて使用することができ、また、アプリの開発も同時に行うことができたので、非常に満足できる実験であった。